Лабораторна робота №4

Вивчення криптосистеми RSA та алгоритму електронного підпису; ознайомлення з методами генерації параметрів для асиметричних криптосистем

Виконала: Годлевська Аліса ФБ-11

Мета роботи

Ознайомлення з тестами перевірки чисел на простоту і методами генерації ключів для асиметричної криптосистеми типу RSA; практичне ознайомлення з системою захисту інформації на основі криптосхеми RSA, організація з використанням цієї системи засекреченого зв'язку й електронного підпису, вивчення протоколу розсилання ключів.

Порядок виконання

1. Написати функцію пошуку випадкового простого числа з заданого інтервалу або заданої довжини, використовуючи датчик випадкових чисел та тести перевірки на простоту. В якості датчика випадкових чисел використовуйте вбудований генератор псевдовипадкових чисел вашої мови програмування. В якості тесту перевірки на простоту рекомендовано використовувати тест Міллера-Рабіна із попередніми пробними діленнями. Тести необхідно реалізовувати власноруч, використання готових реалізацій тестів не дозволяється

2. За допомогою цієї функції згенерувати дві пари простих чисел p, q і p_1 , q_1 довжини щонайменше 256 біт. При цьому пари чисел беруться так, щоб $pq < p_1q_1$; p і q – прості числа для побудови ключів абонента A, p_1 і q_1 – абонента B.

3. Написати функцію генерації ключових пар для RSA. Після генерування функція повинна повертати та/або зберігати секретний ключ (d, p, q) та відкритий ключ (n, e). За допомогою цієї функції побудувати схеми RSA для абонентів A і B – тобто, створити та зберегти для подальшого використання відкриті ключі (e, n), (e_1, n_1) та секретні d і d_1.

4. Написати програму шифрування, розшифрування і створення повідомлення з цифровим підписом для абонентів A і B. Кожна з операцій (шифрування розшифрування, створення цифрового підпису, перевірка цифрового підпису) повинна бути реалізована окремою процедурою, на вхід до якої повинні подаватись лише ті ключові дані, які необхідні для її виконання. За допомогою датчика випадкових чисел вибрати відкрите повідомлення M і знайти криптограму для абонентів A и B, перевірити правильність розшифрування. Скласти для A і B повідомлення з цифровим підписом і перевірити його.

5. За допомогою раніше написаних на попередніх етапах програм організувати роботу протоколу конфіденційного розсилання ключів з підтвердженням справжності по відкритому каналу за допомогою алгоритму RSA. Протоколи роботи кожного учасника (відправника та приймаючого) повинні бути реалізовані у вигляді окремих процедур, на вхід до яких повинні подаватись лише ті ключові дані, які необхідні для виконання. Перевірити роботу програм для випадково обраного ключа $0 < k < n$.

Хід роботи

1-2. Числа, що «не проходять» поміщаються у файл `rejected_primes.txt`

```
def generate_prime_number(bits):
    def get_random_number(bits):
        return random.randint(2 ** bits, 2 ** (bits + 1) - 1)

    def test_miller_rabin(p):
        if p % 2 == 0 or p % 3 == 0 or p % 5 == 0 or p % 7 == 0 or p % 11 ==
0:
            with open("rejected_primes.txt", "a") as file:
                file.write(f"{p}\n")
            return False

        s, d = 0, p - 1

        while d % 2 == 0:
            d //= 2
            s += 1
        assert (p - 1 == d * (2 ** s))
```

```
        x = random.randint(2, p - 2)

        if math.gcd(x, p) > 1:
            with open("rejected_primes.txt", "a") as file:
                file.write(f"{p}\n")
            return False

        if pow(x, d, p) == 1 or pow(x, d, p) == -1:
            return True

        for _ in range(1, s - 1):
            x = (x * x) % p
            if x == -1:
                return True
            if x == 1:
                with open("rejected_primes.txt", "a") as file:
                    file.write(f"{p}\n")
                return False
        with open("rejected_primes.txt", "a") as file:
            file.write(f"{p}\n")
        return False

    num = get_random_number(bits)
    while not test_miller_rabin(num):
        num = get_random_number(bits)

    return num
```

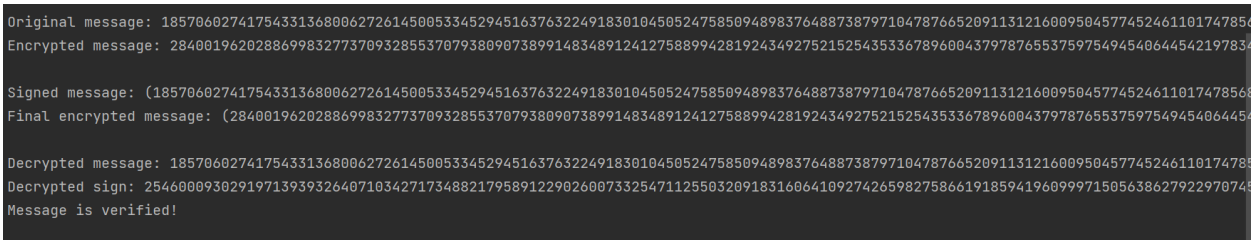
3.

```
Sender:
Public Key: (260430749621558459400664367292488225616986572524175071535979923127684649658600334249582309499517957789027164408391471104012239292877
Private Key: (406837361280729283815860016836366396671606654180463613284917291450819452096482214389440500605649582325912430973086406750764120051

Receiver:
Public Key: (320703340620079670233860736237016688491864514624524063837246534126532138160835336234568363367730602183248377943409678867901367411967
Private Key: (191094895319510066104681258080042398985241942118227706744408721854451783379139111582081661651660247356202188845339102446191226385
```

```
def get_pair(bits):
    pair = (generate_prime_number(bits), generate_prime_number(bits))
    return pair


def generate_keys(pair):
    n = pair[0] * pair[1]
    f = (pair[0] - 1) * (pair[1] - 1)
    e = 2**16 + 1
    d = pow(e, -1, f)
    open_key = (n, e)
    secret_key = (d, pair[0], pair[1])
    return open_key, secret_key
```

Так було створено та збережено для подальшого використання відкриті ключі (e, n), (e_1, n_1) та секретні d і d_1.

4-5.




```python
def encrypt(message, key):
    encrypted_message = pow(message, key[0][1], key[0][0])
    return encrypted_message


def sign(message, key):
    signed_message = (message, pow(message, key[1][0], key[0][0]))
    return signed_message


def decrypt(encrypted, key):
    decrypted_message = pow(encrypted, key[1][0], key[0][0])
    return decrypted_message


def verify(signed, message, key):
    if message == pow(signed, key[0][1], key[0][0]):
        print('Message is verified!')
    else:
        print('Fake sign!')
```

Висновки: під час виконання комп’ютерного практикуму я ознайомилась із методами генерації ключів для асиметричної криптосистеми RSA; практично використала знання: краще ознайомилась із поняттями засекреченого зв'язку й електронного підпису.